Generics

Motivation: Sorting

`Collections` is a static class with a bunch of useful methods that can be employed over `Collection` types.

Here is Collections

And here is Collection, which is different!

Sorting with `Collections`

```
List<Integer> ints = new ArrayList<>();
ints.add(34);
ints.add(-3);
ints.add(23);
ints.add(4020);
System.out.println(ints); // before sorting

Collections.sort(ints);
System.out.println(ints); // after sorting
```

Sorting with `Collections` again

```
List<Double> doubles = new ArrayList<>();
doubles.add(34);
doubles.add(-3);
doubles.add(23);
doubles.add(4020);
System.out.println(doubles); // before sorting

Collections.sort(doubles);
System.out.println(doubles); // after sorting
```

Color

```
public class Color {
    private int r;
    private int g;
    private int b;

    ...
}
```

Pretty simple class!

Sorting with `Collections` again

```
List<Color> colors = new ArrayList<>();
colors.add(new Color(200, 100, 10));
colors.add(new Color(10, 100, 200));
colors.add(new Color(30, 199, 20));
colors.add(new Color(30, 29, 28));
System.out.println(colors); // before sorting

Collections.sort(doubles);
System.out.println(doubles); // after sorting
```

What happens??

Sorting Needs an Order

Objects you write won't automatically have a reasonable order among them.

How do you answer the question of whether one object comes before, after, or is equal to another object?

Sorting with Strings

How do we sort collections of Strings?

Sorting with Strings

How do we sort collections of Strings?

ALPHABETICALLY!

Sorting with Strings

What is alphabetical order defined with?

`compareTo()`.

compareTo() and Comparable

Comparable is an interface with just one method

compareTo()

> Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

- If x is less than y, `x.compareTo(y)` returns a negative value.
- If x is greater than y, `x.compareTo(y)` returns a positive value.
- If x equals y, `x.compareTo(y)` returns `0`.

Exercise:

Reimplement the compareTo method used for `String` objects.

Specifically,

- If x comes before y, `x.compareTo(y)` returns a negative value.
- If x comes after y, `x.compareTo(y)` returns a positive value.
- If x equals y, `x.compareTo(y)` returns `0`.

Tests: "Apple" and "Orange", "Orange" and "Apple", "Apple" and "Apply", ...

How To Make Your Objects Comparable

```
public class YourClass impelments Comparable<YourClass> {

    @Override
    public int compareTo(YourClass otherObj) {
        ...
    }
}
```

How To Make Your Objects Comparable

```
public class Color impelments Comparable<Color> {
    private int r;
    private int g;
    private int b;
    @Override
    public int compareTo(Color otherObj) {
        if (this.r == other.r) {
            if (this.g == other.g) {
                return this.b - other.b;
            } else {
                return this.g - other.g;
            }
        }
        return this.r - other.r;
    }
}
```

What is this `<>` nonsense?

We've seen it with Collections like `List`, `Map`, etc.

We've seen it now with `Comparable` interface.

Used in Generic Methods

```
public static <ThisType extends Comparable<ThisType>>
ThisType minOfThree(ThisType one, ThisType two, ThisType three) {

}
```

`TheType` is any identifier, specifing the **type parameter** that can be used throughout the method for *parameter types, return types, or local var types.*

`extends Comparable<TheType>>` enforces that `TheType` must refer to a type that extends `Comparable`.

Used in Generic Classes

```
public class SortedTriple<ThisType extends Comparable<TheType>> {
    ...
}
```

`TheType` is any identifier, specifing the **type parameter** that can be used throughout the class for *parameter types, return types, field types, or local var types.*

`extends Comparable<TheType>>` enforces that `TheType` must refer to a type that extends `Comparable`.

Collaborative Exercise: Write a Sorted Triple Class

```
public class SortedTriple<ThisType extends Comparable<TheType>> {

}
```

Generics Don't Always Need Bounds

Our friend, the List